图文 95 源码分析的起点：从NameServer的启动脚本开始讲起

详情 评论

源码分析的起点：从NameServer的启动脚本开始讲起

继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、看源码到底难不难？

今天这一讲开始，我们就要正式开始带着大家一步一步的来分析RocketMQ的源码了，这个时候有很多的朋友会感到有点担心，因为大家可能都觉得看源码是一件非常困难的事。

为什么呢？因为很多人平时尝试过自己去看源码，结果看的一头雾水，没法继续看下去。大家也知道，网上其实很多技术都有对应的一些人写了一些源码解析的书籍，有很多人也去买了源码解析的书籍回来看。

结果发现他们写的源码解析的书籍看来看去，似乎还是看不懂，里面的内容极为的晦涩难懂，看了几次就是没法看下去，于是最后又是退缩了。网上也有一些人写的专门讲解源码分析的技术博客，其实有的内容写的还不错，但是很多人发现自己也是看来看去，还是看不懂！

所以因为上述的原因，大部分的Java工程师其实目前都没有能力自己去对一些技术进行源码级别的分析，没有能力去钻研一个技术较为底层的原理。而且很多人对源码这个东西，感觉有一种畏惧感，甚至是恐惧感，觉得自己根本看不懂。

看到这里，大家应该都发现了，看源码难不难？

难！

那么是不是真的大部分人此生都没法看懂一些开源技术的源码了？

未必！我们专栏接下来的这个部分，就是要带着大家一步一步的去看懂RocketMQ的核心源码，我们来换一种方式讲解源码。

2、我们要如何去讲解RocketMQ的源码？

为什么你会看不懂一些书籍或者博客的源码？因为很多人写的源码解析的书籍或者技术博客，往往是站在自己已经理解源码之后的角度去写的。

也就是说，他分析源码的时候，是先分析一个模块的源码，再分析一个模块的源码，接着分析下一个模块的源码。而且分析源码的过程中，以大量的源码和文字描述居多一些，很少有图在里面。

所以这个情况的话，如果是已经读懂这个技术的源码的人，是能看懂这本书的，但是如果是初次看这个技术源码的大多数人，按照这种顺序来，是很难理解的！

写书或者博客的目的，不是应该给那些不懂这个技术的源码的朋友分析吗？因此站在自己已经理解的角度来写文章分析源码，出发点可能就错了

所以我们这个专栏将要采用完全不同于传统的方式来分析RocketMQ的源码，简单来说，可以概括为三点：**场景驱动 + 通俗语言 + 大量画图**

首先，我们分析源码的顺序，不会是先讲NameServer全部的源码，然后是讲Producer全部的源码，接着讲Consumer全部的源码，最后讲Broker全部的源码，我们不会是这种一个模块一个模块的进行源码分析的。

我们会用场景来驱动源码的分析，也就是说，RocketMQ使用的时候，第一个步骤一定是先启动NameServer，那么我们就先来分析NameServer启动这块的源码，然后第二个步骤一定是启动Broker，那么我们再来分析Broker启动的流程。

接着Broker启动之后，必然会把自己注册到NameServer上去，那我们接着分析Broker注册到NameServer这部分源码，然后Broker必然会跟NameServer保持一个心跳，那我们继续分析Broker的心跳的源码。

所以实际上来说，我们分析源码，将会完全按照我们平时使用RocketMQ的各种场景来进行源码的分析，在一个场景中把各种源码串联起来分析，我觉得大家会觉得容易理解的多。

同时在分析的过程中，我们还是尽量用通俗易懂的语言去表达，而且尽量多画一些图给大家，用图来帮助大家理解源码中蕴含的复杂逻辑。

这就是我们将要采取的源码分析思路，那么接下来，我们就一起来开始RocketMQ的源码分析之旅吧。

3、从NameServer的启动开始说起

大家学习了之前的RocketMQ的知识体系之后，肯定都知道一点，那就是RocketMQ要玩儿起来的话，必须是先启动他的NameServer，因为后续Broker启动的时候，都是要向NameServer注册的，然后Producer发送消息的时候，需要从NameServer获取Broker机器信息，才能发送消息到Broker去。

所以我们第一个源码分析的场景，就是NameServer启动这个场景。

那么NameServer启动的时候，是通过哪个脚本来启动的呢？

其实我们之前都给大家讲过了，就是基于rocketmq-master源码中的distribution/bin目录中的mqnamesrv这个脚本来启动的，在这个脚本中有极为关键的一行命令用于启动NameServer进程，如下。

```
sh ${ROCKETMQ_HOME}/bin/runserver.sh org.apache.rocketmq.namesrv.NamesrvStartup $@
```

大家都看到，上面那行命令中用sh命令执行了runserver.sh这个脚本，然后通过这个脚本去启动了NamesrvStartup这个Java类，那么runserver.sh这个脚本中最为关键的启动NamesrvStartup类的命令是什么呢，如下

```
JAVA_OPT="${JAVA_OPT} -server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
JAVA_OPT="${JAVA_OPT} -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -
XX:CMSInitiatingOccupancyFraction=70 -XX:+CMSParallelRemarkEnabled -XX:SoftRefLRUPolicyMSPerMB=0 -
XX:+CMSClassUnloadingEnabled -XX:SurvivorRatio=8  -XX:-UseParNewGC"
JAVA_OPT="${JAVA_OPT} -verbose:gc -Xloggc:${GC_LOG_DIR}/rmq_srv_gc_%p_%t.log -XX:+PrintGCDetails"
JAVA_OPT="${JAVA_OPT} -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=30m"
JAVA_OPT="${JAVA_OPT} -XX:-OmitStackTraceInFastThrow"
JAVA_OPT="${JAVA_OPT} -XX:-UseLargePages"
JAVA_OPT="${JAVA_OPT} -Djava.ext.dirs=${JAVA_HOME}/jre/lib/ext:${BASE_DIR}/lib"
#JAVA_OPT="${JAVA_OPT} -Xdebug -Xrunjdwp:transport=dt_socket,address=9555,server=y,suspend=n"
JAVA_OPT="${JAVA_OPT} ${JAVA_OPT_EXT}"
JAVA_OPT="${JAVA_OPT} -cp ${CLASSPATH}"

$JAVA ${JAVA_OPT} $@
```

大家可以看到，说白了，上述命令大致简化一下就是类似这样的一行命令：

```
java -server -Xms4g -Xmx4g -Xmn2g org.apache.rocketmq.namesrv.NamesrvStartup
```

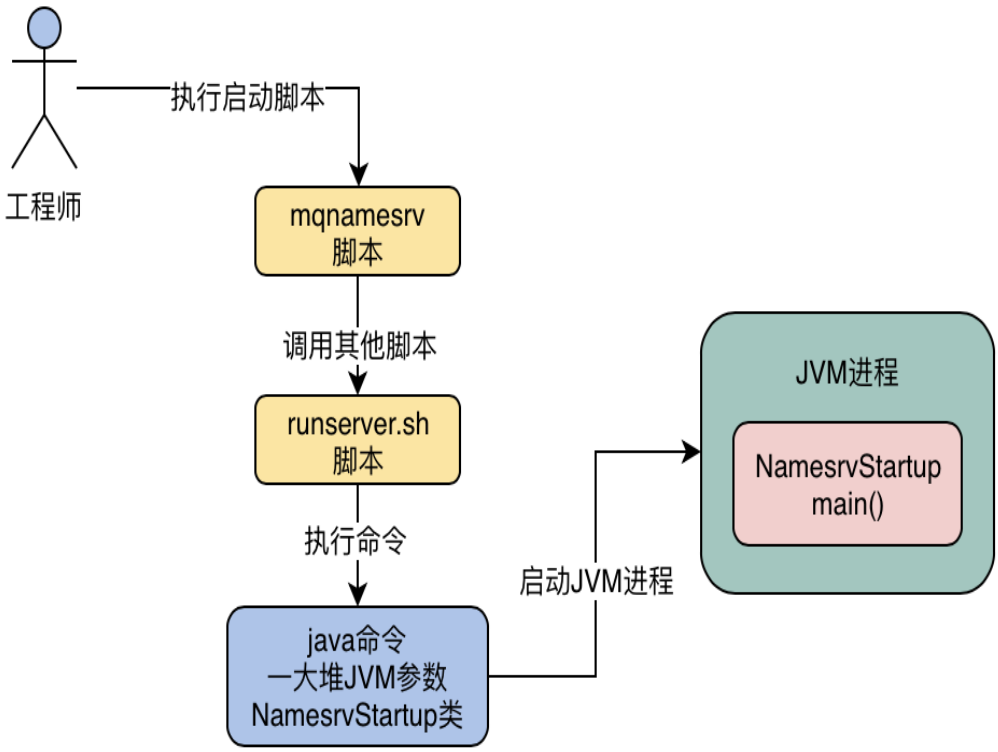
这行命令只要是学习过Java基础的人应该都能理解。

通过java命令 + 一个有main()方法的类，就是会启动一个JVM进程，通过这个JVM进程来执行NamesrvStartup类中的main()方法，这个main()方法里就完成了NameServer启动的所有流程和工作，那么既然NameServer是一个JVM进程，肯定可以设置JVM参数了，所以上面你看到的一大堆-Xms4g之类的东西，都是JVM的参数。

这里给我的好朋友救火队长打个广告，如果大家看不懂上述一大堆JVM参数的话，推荐学习狸猫技术窝里的专栏《从0开始带你成为JVM实战高手》，很高质量的一个专栏

所以说白了，你使用mqnamesrv脚本启动NameServer的时候，本质就是基于java命令启动了一个JVM进程，执行NamesrvStartup类中的main()方法，完成NameServer启动的全部流程和逻辑，同时启动NameServer这个JVM进程的时候，有一大堆的默认JVM参数，你当然可以在这里修改这些JVM参数，甚至进行优化。

这边我也用下面的一张图来展示一下这个启动NameServer的过程，相信大家对照图来看，会理解的更加透彻一些。



4、初步看一眼NamesrvStartup的main()方法

今天要讲的内容其实到这里就结束了，我们先给大家讲一下NameServer是如何通过脚本来启动的，往往源码分析都是从他的启动脚本开始分析的。

但是我们在结束之前，也先来看一眼NamesrvStartup的main()方法，因为明天我们要开始分析他的源码。

```
// 下面这个NamesrvStartup类
// 就是最为关键的NameServer进程的启动类

public class NamesrvStartup {

    // 刚开始看源码，下面的这个InternalLogger、Properties、CommandLine
    // 大家必然是不理解的，不过没关系，这都不是很重要
    // 其实看源码的时候，很多时候刚开始会遇到很多类你都不太理解
    // 但是没必要立马所有的类都要搞明白怎么回事，你可以大致先猜测一下
    // 这几个类肯定是跟日志、配置、命令行参数有关的
    private static InternalLogger log;
    private static Properties properties = null;
    private static CommandLine commandLine = null;

    // NameServer进程启动的时候，必然会执行NamesrvStartup类的main()方法
    public static void main(String[] args) {
        main0(args);
    }

    // 下面这个main0()其实才是真正的main()方法
```

```
21    public static NamesrvController main0(String[] args) {
22
23        // 初步看一眼下面的代码，你会发现里面有NamesrvController之类的东西
24        // 其实有的朋友可能已经猜想到了，这个可能就是NameServer的核心代码组件
25        // 在这里NameServer一启动，就会去启动这个NamesrvController
26        try {
27            NamesrvController controller = createNamesrvController(args);
28            start(controller);
29
30            String tip = "The Name Server boot success. serializeType=" +
31                RemotingCommand.getSerializeTypeConfigInThisServer();
32
33            log.info(tip);
34            System.out.printf("%s%n", tip);
35            return controller;
36
37        } catch (Throwable e) {
38            e.printStackTrace();
39            System.exit(-1);
40        }
41        return null;
42    }
43    // 一大堆省略的源码
44 }
```

好了，我们先大致看一眼上面的那个类，初步的注释和解释我都写在上面了，其实初步看一眼就行，明天我们来继续分析这里面的NameServer启动逻辑。

5、今天给大家留的小作业

今天我给大家留一个课后小作业，希望大家认真去做，大家可以去自己仔细分析一下mqnamesrv脚本中的每一行逻辑，以及runserver.sh中的每一行shell脚本代码的逻辑，透彻的理解一个中间件系统的进程是如何启动起来的。

End

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）